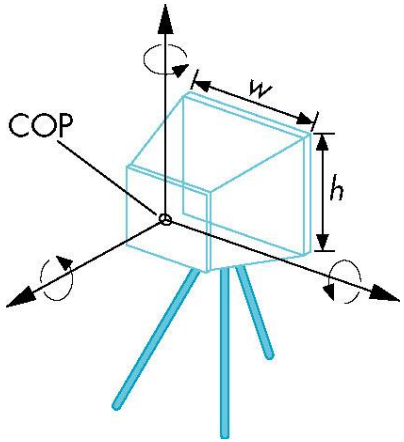
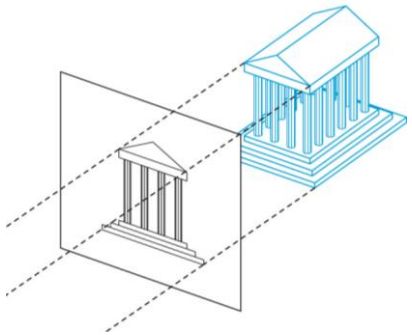
Lecture 3

OpenGl library

Two & three dimensions scenes

- Drawing in two dimensions plane is a special case of 3Ddrawing
  - Viewer at the origin
  - View direction: positive z axis
  - Orthogonal (parallel)projection
  - Specifying the objects in XY (z=0) plane
- OpenGL drawing includes
  - Points
  - Line segments
  - Polygons
- Each primitive is completely specified by points in the 3D (called vertices)
- How we can specify a vertex?





Vertex specification

- glVertex***();
- The first * could be 2 (z=0), 3(3D) or 4 (Homogeneous) to denote the number of coordinates
- The second star could be i for integer, f for float, or d for double
- The third star could be v for passing coordinates in an array or null for giving the coordinates directly
- Instead of using c standard types use GL types
- Use Glint instead of int
- Use Glfloat instead of float
- Use Gldouble instead of double, an so on

```
glVertex3f(x,y,0);
glVertex2f(x,y);
```

```
glVertex2i(GLint xi, GLint yi)
glVertex3f(GLfloat x, GLfloat y, GLfloat z)
```

```
GLfloat vertex[3]
glVertex3fv(vertex)
```

```
glBegin(GL_LINES);
  glVertex3f(x1,y1,z1);
  glVertex3f(x2,y2,z2);
glEnd();
```

```
glBegin(GL_POINTS);
  glVertex3f(x1,y1,z1);
  glVertex3f(x2,y2,z2);
glEnd();
```

OpenGL function classification

- Primitive functions
- Attribute functions
- Viewing functions
- Transformation functions
- Input functions
- Control functions
- Query functions

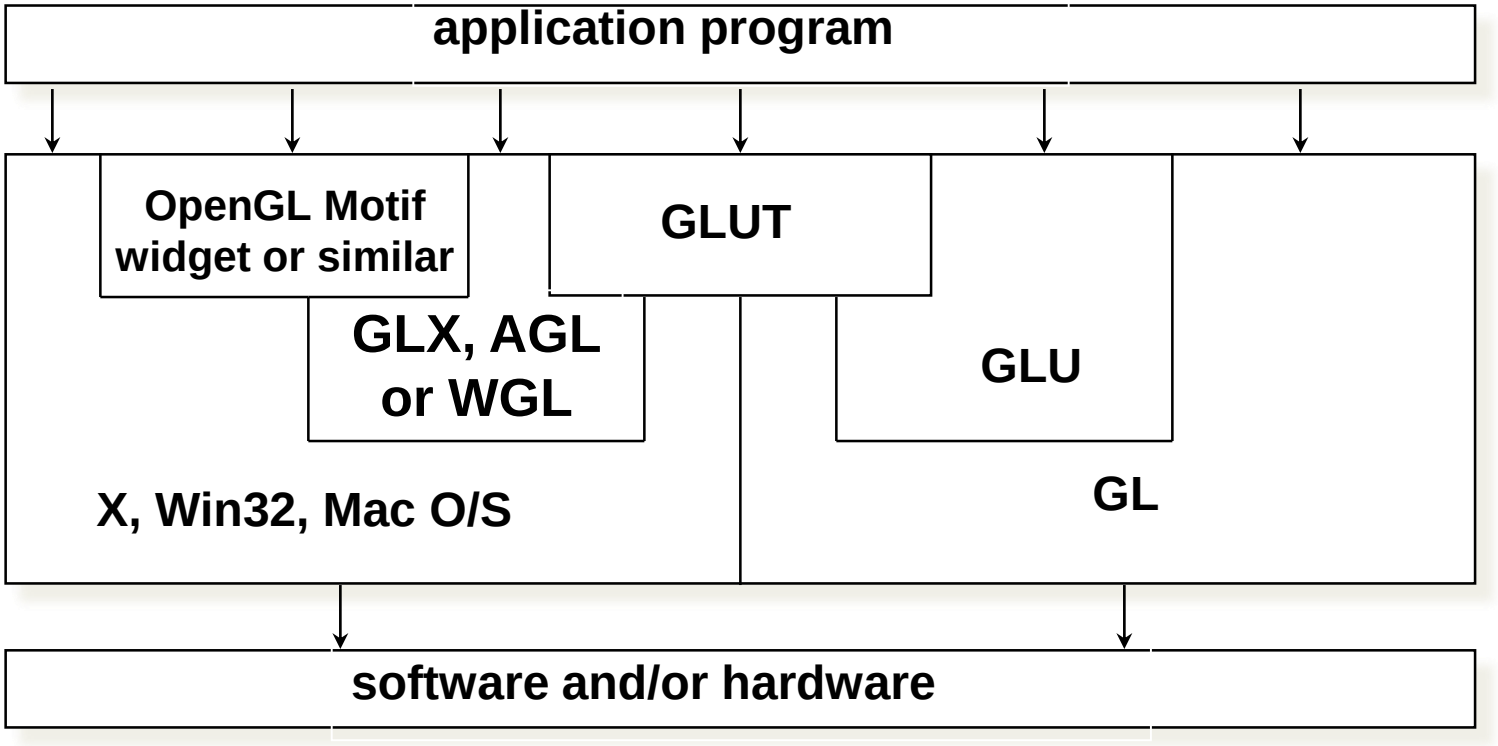
OpenGL libraries

- OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)
- OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code (use OpenGL to provide complex tasks)
- Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh

GLUT

- OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
    - No slide bars

Software organization



OpenGL as a state machine

- OpenGL is a state machine
- OpenGL functions are of two types
  - Primitive generating
    - Can cause output if primitive is visible
    - How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    - Transformation functions
    - Attribute functions
- Not object oriented (line color is not the line property but is a OpenGL state variable

Report Discussion

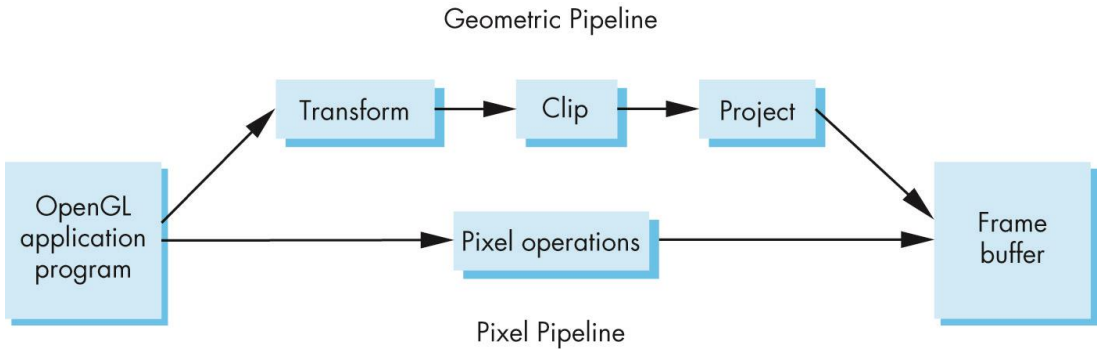
Previous lecture repot:
Problem 1.7

Next lecture repot:
Problem 2.3

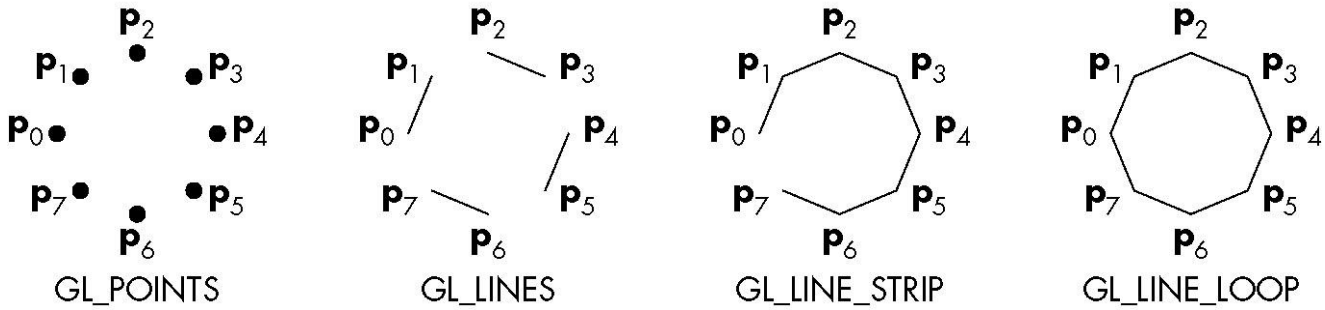
At the lowest level of processing, we manipulate bits in the frame buffer. OpenGL has pixel-oriented commands that allow users to access the frame buffer directly. You can experiment with simple raster algorithms, such as drawing lines or circles, by using the OpenGL function `glPoint` as the basis of a simple virtual-frame-buffer library. Write a library that will allow you to work in a frame buffer that you create in memory. The core functions should be `WritePixel` and `ReadPixel`. Your library should allow you to set up and display your frame buffer and to run a user program that reads and writes pixels.

Primitives: Points and Lines

```
glBegin(type);
    glVertex*(...);
    .
    .
    .
    glVertex*(...);
glEnd();
```



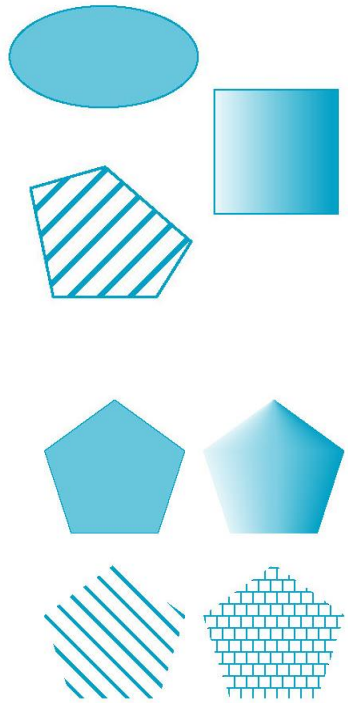
Type



Primitives: Polygons

Polygon: An interior defined by a boundary (vertices)
Boundary can be drawn or not
The interior can be filled with a solid color, shades, or patterns
The function glPolygonMode((GLenum face, Glenum mode);)

- Face:
  - GL_FRONT
  - GL_BACK
  - GL_FRONT_AND_BACK
- Mode:
  - GL_POINT
  - GL_FILL
  - GL_LINE

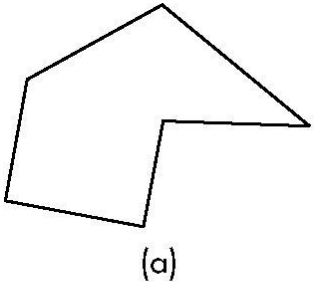


Primitives: Triangles as special polygon in graphics system

- Three properties will ensure that a polygon will be displayed correctly: It must be simple, convex, and flat.
  - Simple: as long as no two edges of a polygon cross each other, we have a simple polygon. Testing is very complex and is left to the application program not to the gl lib.
  - Convex: An object is convex if all points on the line segment between any two points inside the object, or on its boundary, are inside the object. Testing is very complex and is left to the application program not to the gl lib.
  - Flat: the entire object lies in the same plane

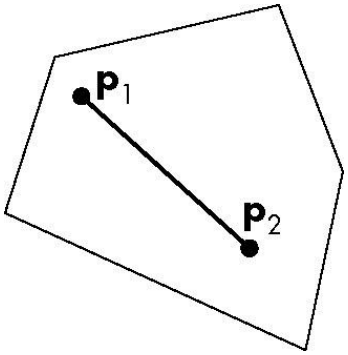
Three vertices that are not collinear determine both a triangle and the plane in which that triangle lies. Hence, if we always use triangles, we are safe(simple, convex and flat)—we can be sure that these objects will be rendered correctly.

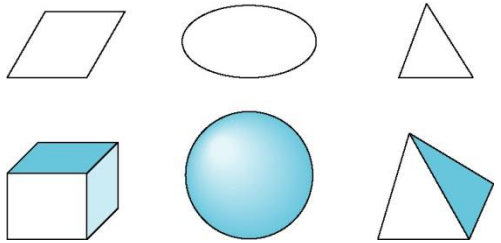
Primitives: Simple and convex polygons



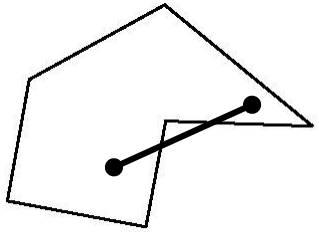
(a) simple, (b) non simple

(a)





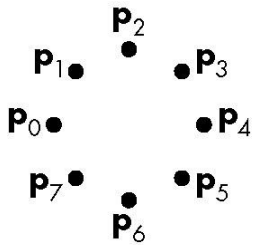
(b)



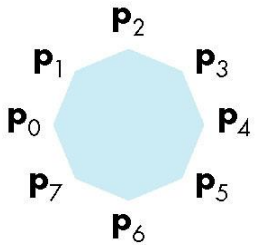
Convex polygon and objects (a) and non-convex polygon (b)

Primitives: Polygons types in openGL

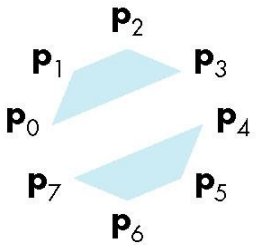
```
glBegin(type);
    glVertex*(...);
    .
    .
    .
    glVertex*(...);
glEnd();
```



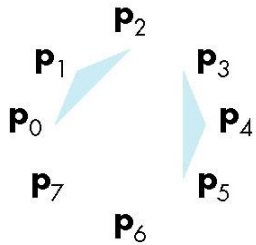
GL_POINTS



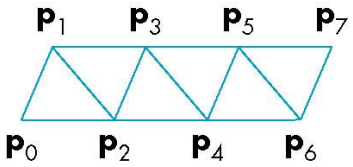
GL_POLYGON



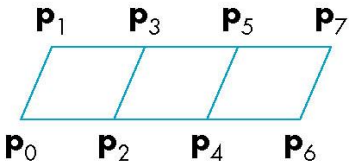
GL_QUADS



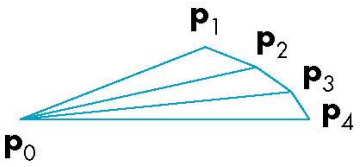
GL_TRIANGLES



GL_TRIANGLE_STRIP

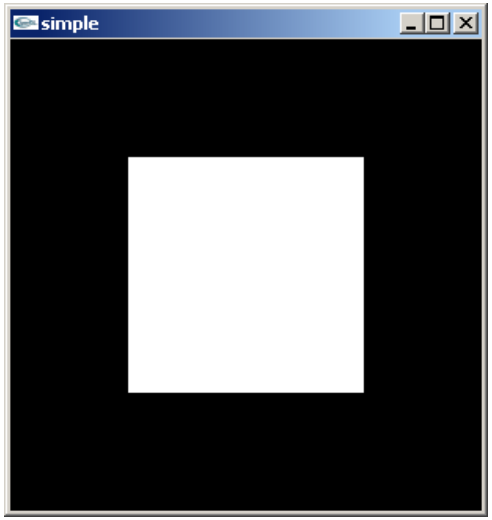


GL_QUAD_STRIP



GL_TRIANGLE_FAN

Simple drawing




```
#include <GL/glut.h>
void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
            glVertex2f(-0.5, -0.5);
            glVertex2f(-0.5, 0.5);
            glVertex2f(0.5, 0.5);
            glVertex2f(0.5, -0.5);
        glEnd();
        glFlush();
}
int main(int argc, char** argv){
        glutCreateWindow("simple");
        glutDisplayFunc(mydisplay);
        glutMainLoop();
}
```

Execution of openGL programs

- Note that the program defines a *display callback* function named **`mydisplay`**
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The **`main`** function ends with the program entering an event loop: **glutMainLoop();**